DYNAMIC ADAPTIVE MULTIMESH REFINEMENT FOR COUPLED PHYSICS EQUATIONS APPLICABLE TO NUCLEAR ENGINEERING

A Thesis

by

KEVIN JAMES DUGAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jean Ragusa
Committee Members,	Ryan McClarren
	Andrea Bonito
Department Head,	Yassin Hassan

August 2013

Major Subject: Nuclear Engineering

ABSTRACT

The processes studied by nuclear engineers generally include coupled physics phenomena (Thermal-Hydraulics, Neutronics, Material Mechanics, etc.) and modeling such multiphysics processes numerically can be computationally intensive. A way to reduce the computational burden is to use spatial meshes that are optimally suited for a specific solution; such meshes are obtained through a process known as *Adaptive Mesh Refinement* (AMR). AMR can be especially useful for modeling multiphysics phenomena by allowing each solution component to be computed on an independent mesh (Multimesh AMR). Using AMR on time dependent problems requires the spatial mesh to change in time as the solution changes in time. Current algorithms presented in the literature address this concern by adapting the spatial mesh at every time step, which can be inefficient. This Thesis proposes an algorithm for saving computational resources by using a spatially adapted mesh for multiple time steps, and only adapting the spatial mesh when the solution has changed significantly. This Thesis explores the mechanisms used to determine when and where to spatially adapt for time dependent, coupled physics problems. The algorithm is implemented using the Deal.ii finite element library [1, 2], in 2D and 3D, and is tested on a coupled neutronics and heat conduction problem in 2D. The algorithm is shown to perform better than a uniformly refined static mesh and, in some cases, a mesh that is spatially adapted at every time step.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor, Dr. Jean Ragusa, for dedicating the time necessary to guide me through this project. I would also like to thank Dr. Ragusa for his assistance in preparing a topic for a Ph.D. dissertation to be carried out at the CEA-Saclay, France. I would also like to acknowledge my committee members, Dr. Ryan McClarren and Dr. Andrea Bonito, for the time they have dedicated in helping this project succeed.

I would like to thank my family for being a constant stream of encouragement and inspiration. I would like to thank my friends Annabelle, Randall, Marco, and Emma for making my Master's study an enjoyable and memorable experience. I am grateful that they are genuinely interested in what my research entails. Thank you to everyone who has supported me in my goals.

NOMENCLATURE

FEM	Finite Element Method
PDE	Partial Differential Equation
MMS	Method of Manufactured Solutions
AMR	Adaptive Mesh Refinement
Deal.ii	Differential Equations Analysis Library
DOF	Degree Of Freedom

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Many of the physical processes occurring in the analysis of nuclear systems are strongly coupled. For example, fuel temperature affects the rate of fission in the fuel and the fission rate affects the temperature field — mathematically this manifests through the temperature dependence of the macroscopic cross sections. When modeling nuclear systems these interactions present themselves as nonlinear coupling terms in the governing equations, which can be challenging to solve numerically.

Additionally, solving for the analytical solution to these equations is not a realistic goal. The alternative is to solve a discrete approximation to the continuous problem; however, this introduces numerical error. The amount of error introduced can be reduced by decreasing the size of the spatial mesh used to approximate the solution. An effective way to generate a mesh that is optimized to the specific solution being sought is to use Adaptive Mesh Refinement (AMR). In this process, the spatial mesh is locally refined based on the solution's properties at that location and left unrefined in areas where refinement is not needed.

When modeling multiphysics phenomena, it is likely that the multiple components of the solution will have very different smoothness. It would be unreasonable to expect that a single mesh produced through AMR would be optimal for all solution components. A technology which has shown potential in multiphysics modeling is Multimesh AMR, where each solution component is allowed to have an independent mesh that is adapted to its features. This allows for each solution component to be computed on a spatial mesh that is optimal for that component.

A consequence of using AMR on time dependent problems is that the spatial mesh is required to change with the solution in time. Current algorithms address dynamic

mesh refinement by spatially adapting the mesh at every time step; additionally, these algorithms may start the refinement process on a coarse mesh at every time step. This strategy can be useful when the solution is changing rapidly in time, but may be inefficient if the solution varies slowly in time. A method for using a spatial mesh for multiple time steps is developed and discussed in this Thesis.

The topics covered in this Thesis can improve how effectively computational resources are used while seeking the solution to a time-dependent problem. The savings in resources can be used to either solve a problem more accurately, or reach a solution quicker. In short, this Master's Thesis is a study of how to efficiently and accurately solve nonlinear, coupled, unsteady equations using Multimesh AMR technology.

1.1 Motivation of Adaptively Refined Meshes

The theory of numerical solutions to partial differential equations states that as the size of the mesh decreases, the error between the exact solution and the numerical solution also decreases at prescribed rates [8]. This means that in order to obtain a more accurate solution, the number of spatial cells, and thus the number of unknowns, should increase.

Consequently, when the number of unknowns increases, the wall-clock time to obtain a solution also increases. Solving large numerical systems is limited by how much computing power is present, the more computing power the larger the system size can be solved. What is desirable is a way to achieve the same level of numerical error with a smaller system. This can be done by choosing where to locally refine a spatial mesh to obtain the maximum reduction in error. This is the motivation behind Adaptive Mesh Refinement — to achieve an accurate solution with fewer unknowns by locally refining a mesh rather than using a uniformly refined mesh.

If a solution has strong local features, it is likely that the solution error is concentrated in these locations of the domain; these local error contributions can dominate the solution error. Thus, if the cells in these locations are refined, then the dominant source of solution error can be reduced. If the mesh is not adapted in regions where the local error contribution is small, the system size, and thus the number of unknowns, will remain small. Furthermore, since the dominant source of error is being reduced without refining the mesh uniformly, the same level of error can be achieved with fewer unknowns.

Furthermore, when modeling multiple physical processes simultaneously, as is the case for many modern algorithms [6, 13, 19, 20], each physical process can be computed on an independent mesh. The illustration given in [19] describes the different properties of a temperature field and a displacement field. The temperature field in a material is generally smooth, contrary to the displacement field which has singularities at re-entrant corners. Singular points require high levels of refinement to properly resolve the solution gradients; if the temperature field is required to use the same mesh as the displacement field, there will be far too much refinement for the temperature than is needed in these areas. Using independent meshes allows each solution to have a mesh that is optimized to its properties without influence from other solutions.

1.2 The State-of-the-art in Nuclear Engineering

The nuclear engineering field encompasses the study of many physical processes (particle transport, radiative heat transfer, thermo-mechanical stresses, etc.), each with some type of multiphysics phenomenon influencing the solution process. The accurate solution to these types of problems allow scientists and engineers to design and build better power generation stations, understand the inter workings of

supernovae, and more [5, 15, 24]. In all of these cases, the numerical systems can be extremely large. The advantage of using AMR technology is that it can achieve the same level of solution error with a smaller problem to solve and thus it can be a worthwhile technology to study and incorporate in the nuclear engineering field.

Adaptive Mesh Refinement has slowly emerged in the nuclear engineering field in the last few years [13, 21, 22]. This technology has been used to solve for the criticality state of reactors modeled in two and three dimensions [21, 22]. In these papers, the neutron diffusion equations with two energy groups were solved using Multimesh AMR; each energy group had an independent mesh with coupling between the two groups through neutron scattering and fission. Multimesh AMR is helpful in resolving the solution shape accurately, which is necessary to determine the criticality state of a nuclear system.

AMR has also been used to solve coupled physics problems, in particular the coupling between the neutron flux and the temperature field [13]. That study solves the time dependent, coupled, heat conduction and neutron diffusion equations and builds on the techniques developed in [6, 20]. The method used required that the spatial mesh be adapted at every time step. This can yield accurate results for problems changing drastically through the time evolution, but might be inefficient for problems that are not changing drastically at every time step.

1.3 Improvements to the State-of-the-art

The studies [6, 13, 20] can be considered what is the present state-of-the-art in multiphysics simulations applied to nuclear engineering; this Thesis extends directly from the methods described in these papers. Improvements made to the current state-of-the-art will address the use of spatial meshes for multiple time steps and the use of higher order time integrators.

In the studies [6, 13, 20], the spatial mesh was refined at every time step. This constraint can be useful for cases when a solution is changing rapidly, but can be excessive if the solution is slowly varying in time. An improvement can be made by allowing a given spatial mesh to be used for several time steps, and to only be adapted when the solution has changed by a significant amount. An algorithm is explored that quantifies how much a solution changes in time and spatially adapts when the solution has changed significantly.

The authors of [13] remarked that using first order time discretization methods required the time step size to be small. The implementation of higher order time discretization methods allows for larger time steps to be taken while still maintaining a low solution error. A second order method was used in [20], but the implementation was fixed for that specific method. A general way to implement Runge-Kutta methods is explored in this Thesis.

The following outlines the structure for the remainder of the Thesis. Chapter 2 introduces numerical methods that will be useful to understand how the numerical systems discussed are built and solved. Special emphasis will be given to the complexities that arise when coupling terms are present between physical models and it is desired to solve each variable on an independent mesh. The end of Chapter 2 discusses the proposed algorithm for utilizing a spatial mesh for multiple time steps. Once the numerical tools have been introduced, the process of code verification is introduced in Chapter 3. Several exact solutions are considered to test various properties of the proposed algorithm. The Thesis ends with conclusions about the performance of the proposed algorithm and outlines possible extensions to this project.

2. SOLUTION TECHNIQUES

The nuclear engineering field encompasses the understanding of many physical processes, which require many techniques to accurately solve equations that model these processes. A specific facet of the nuclear engineering field, namely reactor dynamics, will be explored to develop solution techniques that can serve a much broader purpose. This chapter details the methods used in this Thesis to accurately solve this model problem relevant to nuclear engineering, which will be a coupled, nonlinear, multiphysics model of neutron diffusion and heat conduction. The foundation of the solution techniques described in this chapter build on the formation of nonlinear residuals and Newton's Method, thus the first section of this chapter will introduce these concepts. The model problem is a set of time dependent partial differential equations which will require spatial and temporal discretization, thus the second and third sections will address discretization. The remainder of the chapter is dedicated to improvements made on the state-of-the-art in nuclear engineering.

Now is a good point to introduce the model problem studied during this Thesis. Equation 2.1 represents the one group neutron diffusion equation and the heat conduction equation. This model is simplified from reality, however it can be used to develop the solution techniques necessary for many types of multiphysics problems.

$$\begin{bmatrix} \frac{1}{v} & 0 \\ 0 & \rho C_p \end{bmatrix} \frac{\partial}{\partial t} \begin{bmatrix} \phi \\ T \end{bmatrix} = \begin{bmatrix} \nabla \cdot D \nabla - \Sigma_a(T) + \nu(1-\beta)\Sigma_f & 0 \\ \kappa & \nabla \cdot k \nabla \end{bmatrix} \begin{bmatrix} \phi \\ T \end{bmatrix} + \begin{bmatrix} Q_\phi \\ Q_T \end{bmatrix} \quad (2.1)$$

The coefficient matrix in front of the time derivative will later be referred to as G; the coefficients $\frac{1}{v}$ and ρC_p are the inverse of the neutron velocity and the

material heat capacity. The diffusion term in the neutronics equation, $\nabla D \nabla$, and the same term in the heat equation, $\nabla k \nabla$, are not dependent on the temperature as is in the most general case. The term $\Sigma_a(T)$ is the absorption coefficient that has a nonlinear dependence on temperature; the exact form of the absorption cross section's dependence on temperature is not important at this point. The term $\nu(1-\beta)\Sigma_f$ is the fission cross section. Notice that the coupling term κ provides a linear dependence of temperature on the neutron flux. A more compact way of writing Equation 2.1 is given by denoting the solution vector $\vec{U} = [\phi \quad T]^T$ and making the right hand side be the steady state residual.

$$\frac{\partial \vec{U}}{\partial t} = G^{-1}\tilde{f}(\vec{U}, t) \tag{2.2}$$

Here G is the diagonal matrix of coefficients that appear in front of the time derivatives in Equation 2.1; for the rest of the discussion, the coefficient matrix G will be incorporated in the residual $f = G^{-1}\tilde{f}$. Equations 2.1 & 2.2 contains many challenging aspects — tight couplings, nonlinear coefficients, etc. A variety of mathematical techniques will be needed to solve such a system of equations, which will be introduced in the remainder of this chapter.

2.1 Newton's Method

As stated in the introduction of this chapter, many of the physical processes that are of interest to nuclear engineers are nonlinear. This means that any methodology for solving equations related to nuclear engineering must include a way to resolve such nonlinear dependencies. Newton's method is an extensively studied method for solving nonlinear equations [9, 10, 12, 14] and is the choice of nonlinear solver for this Thesis.

Newton's method involves finding the roots of the nonlinear residual function

$[F(\vec{U}) = 0]$ formed after spatial and temporal discretization. Newton's method locates the roots of $F(\vec{U})$ by calculating the tangent plane of the residual function at the current iterate and locating the roots of this linear approximation of the residual function [10]. The sequence of a Newton iteration is given by

$$J(\vec{U}^l)\delta\vec{U} = -F(\vec{U}^l),$$
$$\vec{U}^{l+1} = \vec{U}^l + \delta\vec{U},$$

where the Jacobian matrix $J(\vec{U}^l)$ is formed by differentiating the residual with respect to every unknown. The iteration process is terminated when the nonlinear residual is sufficiently small when compared to the initial residual.

2.1.1 Formulation of Residual and Jacobian

From Equation 2.1, the nonlinear residual can be formed by moving all terms to one side of the equality sign. Thus the nonlinear residual would have the form

$$F(\vec{U}) = \begin{bmatrix} F^{\phi} \\ \\ F^{T} \end{bmatrix}, \tag{2.3}$$

where the residual is comprised of the residual from each equation in 2.1; the order in which individual residuals are placed in the system is arbitrary.

The Jacobian matrix can be computed analytically or approximated numerically. For the Jacobian to be computed analytically, one needs access to the functional form of the residual and also needs to store the entries of the Jacobian matrix. In many applications either access to the governing equations or the storage size of the Jacobian is prohibitive; in such cases it is necessary to approximate the Jacobian. Using

a direct linear solver requires the Jacobian to be approximated and stored, while an iterative Krylov solver allows the Jacobian to be approximated without storage [10]. In this Thesis, the Jacobian matrix is computed analytically because the governing equations and the functional form of the parameter and closure relationships are well known and can be easily differentiated; this is also an attempt to reduce the possibility of numerical errors. The storage size of the Jacobian may become prohibitive if more accurate solutions are desired; the use of low-storage methods (JFNK) can be considered for the future work of this project. To form the Jacobian matrix, each equation is differentiated with respect to each unknown variable. The Jacobian matrix takes the form

$$J(\vec{U}) = \begin{bmatrix} \frac{\partial F^{\phi}}{\partial \phi} & \frac{\partial F^{\phi}}{\partial T} \\ \frac{\partial F^{T}}{\partial \phi} & \frac{\partial F^{T}}{\partial T} \end{bmatrix} = \begin{bmatrix} J_{\phi\phi} & J_{\phi T} \\ J_{T\phi} & J_{TT} \end{bmatrix}, \tag{2.4}$$

where the second notation is introduced for simplicity. The Jacobian in this form is a block sparse matrix and will require linear solvers. Since the problem sizes studied in this Thesis are relatively small, direct linear solvers will be used.

2.1.2 Linear Solves

Once the nonlinear system is formed, the linear system must be solved at every Newton iteration. Since the decision was made to analytically compute the Jacobian matrix, there is no restriction on the type of linear solver used. Early in this study, both GMRes and a direct linear solver were implemented. However, it was discovered that the linear system was small enough and this meant that a direct method was more efficient than a non-stationary iterative method. If the problem size grows, it is likely that the non-stationary iterative method will become more efficient either because the size of the system is large or that a Jacobian-free method

will be implemented because of memory usage constraints.

2.2 Temporal Discretization

In time-dependent problems, there needs to be an implementation of some time marching method. In general these methods start with an initial condition and approximate the time derivative using a numerical integration approximation. Time dependent problems can be written in standard form given by Equation 2.5. For this Thesis, the forcing term on the right hand side is the steady state residual as seen in Equation 2.2.

$$\frac{\partial \vec{U}}{\partial t} = f(\vec{U}, t) \tag{2.5}$$

The time dependent problems applicable to reactor analysis are usually stiff problems, which require an implicit type of time integrator to solve the problem with reasonable time step sizes. Runge-Kutta methods are chosen for the implementation of this Thesis because a general framework can be implemented that can handle multiple time integrators of various orders. Special focus is given to "Singly Diagonally Implicit Runge-Kutta" (SDIRK) methods because the stages of these methods are able to be solved sequentially rather than concurrently in one system [4]. The Y-formulation for a general s-stage Runge-Kutta Method will be defined as

$$\begin{aligned} \vec{Y}_i &= \vec{U}^n + \tau \sum_{j=1}^{i} a_{ij} f(\vec{Y}_j, t_n + c_j \tau), \qquad i = 1, 2, ..., s \\ \vec{U}^{n+1} &= \vec{U}^n + \tau \sum_{i=1}^{s} b_i f(\vec{Y}_i, t_n + c_i \tau), \end{aligned} \tag{2.6}$$

where τ is the time step size and $\{a, b, c\}$ are given by the Butcher Tableaux characteristic of the Runge-Kutta method used.

In general, the sum index in the stage-i residual ranges from 1 to s; then the

entire classification of the Runge-Kutta method is dependent on the form of A. If A is strictly lower triangular (non-zero for $i < j$), the Runge-Kutta method is explicit and a linear system can be solved at every stage. In the case of A being lower triangular (non-zero for $i \leq j$), the method is said to be "diagonally" implicit and a nonlinear solve is needed at every stage, but the stages can be computed sequentially. In the case that A is full, the method is fully implicit and a nonlinear solve for all stages must be accomplished concurrently. Since this Thesis focuses on SDIRK methods, the sum index on the stage-i is restricted to $j \in [1, i]$, and the Butcher Tableaux reflect this structure. A general Butcher Tableau has the form

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

where the vectors b, c are of size s and the matrix A is of size $s \times s$. An explanation of the Butcher Tableaux for the time integrators used in this thesis are provided in Appendix A.

2.2.1 Transient Residual Formulation

The transient residual is formed using Rothe's method of discretizing the time derivative and then discretizing the spatial domain. The treatment of spatial discretization is detailed in the next section. The formation of the transient residual before spatial discretization becomes Equation 2.7.

$$F(Y_i) = Y_i - \vec{U}^n + \tau \sum_{j=1}^{i} a_{ij} f(Y_j, t_n + c_j \tau), \qquad i = 1, 2, ..., s \tag{2.7}$$

After the solution of all stage residuals, a linear combination of the stage residuals is added to the current solution to yield the solution at the next time step Equation 2.8.

$$\vec{U}^{n+1} = \vec{U}^n + \tau \sum_{i=1}^{s} b_i f(Y_i, t_n + c_i \tau) \tag{2.8}$$

2.2.2 Low Order Runge-Kutta Methods

The reason a general s-stage Runge-Kutta method is used in this Thesis is to provide a framework that allows a variety of time integrators to easily be implemented. The restriction imposed is that the method needs to be diagonally implicit. Various Butcher Tableaux implemented in this Thesis are given in Appendix A. The convergence order of some methods are shown in Table 2.1.

Table 2.1: Outline of Time Marching Methods

Backward Euler (BE)	$\mathcal{O}(\tau)$
Crank-Nicholson (CN)	$\mathcal{O}(\tau^2)$
SDIRK22	$\mathcal{O}(\tau^2)$
SDIRK33	$\mathcal{O}(\tau^3)$

While higher order methods could easily be implemented, the goal of this aspect of the Thesis is to outline the framework for which additional methods could be used. To implement higher order Runge-Kutta methods, one simply needs to input the Butcher Tableau characteristic for that method.

2.3 Spatial Discretization

2.3.1 Continuous Galerkin FEM

The Finite Element Method has been used to numerically solve many types of partial differential equations [18]. The method involves transforming the PDEs into

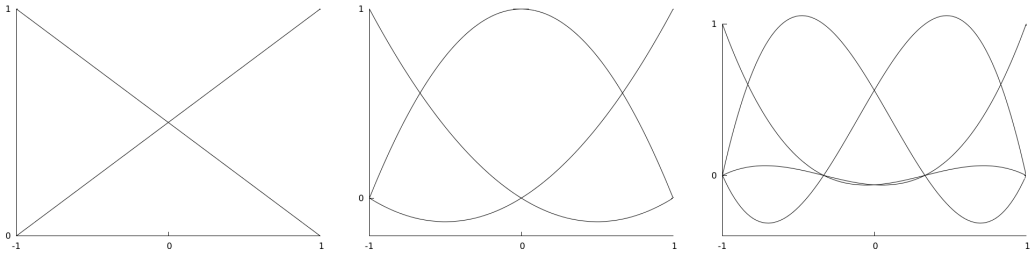
their respective weakforms by multiplying by a test function, denoted by b_i, and integrating over the entire domain Ω. The solution can then be represented as an infinite series of shape functions given by

$$U(\vec{x}) = \sum_{j=1}^{\infty} b_j(\vec{x}) U_j, \tag{2.9}$$

where the $\{U_j\}$ are coefficients of the expansion. No approximations have been made thus far, and if the coefficients $\{U_j\}$ could be computed, the exact solution would be produced. It is unrealistic to solve for an infinite amount of unknowns on a finite precision machine and so a truncated sum approximation is made.

$$U(\vec{x}) = \sum_{j=1}^{N} b_j(\vec{x}) U_j \tag{2.10}$$

The next choice to make is choosing the form of the shape functions and test functions. In the Continuous Galerkin Method, the shape functions and test functions are chosen to be the same [18]; in general these functions are chosen to be local polynomials over a mesh cell. This means that these functions are only non-zero over a few cells in the mesh and zero everywhere else. An example of the first three orders of such functions over the unit cell in one dimension [7] are given by Figure 2.1.



(a) Linear Order (b) Quadratic Order (c) Cubic Order

Figure 2.1: Lagrangian finite element basis functions on the unit cell in one dimension for three polynomial orders

Regardless of the basis functions used, the mathematics for the finite element method are the same. It is illustrative to take a simple example such as a simple diffusion equation in a linear system.

2.3.2 Discretization Formulation

The Diffusion equation is given by Equation 2.11; it is simply the Laplacian of the solution with an interaction term and a forcing term.

$$-\nabla \cdot D(\vec{x})\nabla U(\vec{x}) + \Sigma_t(\vec{x})U(\vec{x}) = q(\vec{x}) \qquad \vec{x} \in \Omega \tag{2.11}$$

Equation 2.11 can be transformed to its weak form by multiplying by a test function and integrating over the domain Ω,

$$-\int_\Omega d\vec{x}\, b_i(\vec{x})\nabla \cdot D(\vec{x})\nabla U(\vec{x}) + \int_\Omega d\vec{x}\, b_i(\vec{x})\Sigma_t(\vec{x})U(\vec{x}) = \int_\Omega d\vec{x}\, b_i(\vec{x})q(\vec{x})$$

$$-\left(b_i(\vec{x}), \nabla \cdot D(\vec{x})\nabla U(\vec{x})\right)_\Omega + \left(b_i(\vec{x}), \Sigma_t(\vec{x})U(\vec{x})\right)_\Omega = \left(b_i(\vec{x}), q(\vec{x})\right)_\Omega \tag{2.12}$$

Here the convenient inner product notation is used to denote integration over the

domain. The first term of Equation 2.12 can be integrated by parts to transform the Laplacian operator to a gradient operator (the test function's spatial dependence designation is removed from Equation 2.13 to write the equation more compactly).

$$(\nabla b_i, D(\vec{x})\nabla U(\vec{x}))_\Omega - (b_i\vec{n}, D(\vec{x})\nabla U(\vec{x}))_{\partial\Omega} + (b_i, \Sigma_t(\vec{x})U(\vec{x}))_\Omega = (b_i, q(\vec{x}))_\Omega \quad (2.13)$$

The boundary term is generally carried through the derivation and will influence the solution. However in the applications discussed in this Thesis, the boundary conditions are either imposed (Dirichlet) or the gradient of the solution is zero (Reflecting); thus the boundary term will evaluate to zero in either case and will be dropped from this example.

The linear combination of basis functions approximation for the solution is inserted into the weakform. The terms of the weakform can be represented by dot products with the unknown coefficients and can be transformed into a linear system.

$$A\vec{U} = \vec{Q} \quad (2.14)$$

Here $A_{ij} = (\nabla b_i(\vec{x}), D(\vec{x})\nabla b_j(\vec{x}))_\Omega + (b_i(\vec{x}), \Sigma_t(\vec{x})b_j(\vec{x}))_\Omega$ and $Q_i = (b_i(\vec{x}), q(\vec{x}))_\Omega$. These terms show up frequently in the finite element method and thus are given special names. The product of the gradient of two basis functions integrated over the domain is called the "stiffness matrix", and is denoted "K_D". The product of two basis functions integrated over the domain is called the "mass matrix", and is denoted by "M_Σ". The subscripts on the special matrices denote the dependence on spatially varying coefficients. This equation can be solved by inverting the matrix $A = K_D + M_\Sigma$. In this way, the mass and stiffness matrices can be computed for

any order of basis functions and the resulting system is similar apart from its size and the values of the entries.

In practice, the integration over the domain cannot be carried out analytically and must be approximated by numerical quadrature. In addition, the domain is broken into a spatial mesh, denoted $\mathcal{T}$, which contains non-overlapping cells that cover the entire domain; in the Deal.ii library, the cells are regular quadrilaterals in 2D and hexahedra in 3D. The integration over the entire domain can be broken into a sum of integrations over cells. For instance to evaluate the A_{ij} matrix, the following transformation is performed.

$$A_{ij} = \sum_{K \in \mathcal{T}} \int_K d\vec{x}\ \nabla b_i(\vec{x}) D(\vec{x}) \nabla b_j(\vec{x}) + b_i(\vec{x}) \Sigma_t(\vec{x}) b_j(\vec{x}) \tag{2.15}$$

To proceed further however, a numerical approximation must be made to perform the integral over a cell; this numerical approximation is known as quadrature integration the details of which do not add to the understanding of this topic.

2.3.3 Coupled Physics

This Thesis is oriented towards solving coupled physics problems which can add complexities to spatial discretization. Taking the steady state residual from Equation 2.1 and applying the techniques developed in the previous section produces a discretized steady state residual Equation 2.16.

$$F^{SS} = \begin{bmatrix} K_D^{\phi\phi} + M_\Sigma^{\phi\phi} & 0 \\ M_\kappa^{T\phi} & K_k^{TT} \end{bmatrix} \begin{bmatrix} \phi \\ T \end{bmatrix} + \begin{bmatrix} Q_\phi \\ Q_T \end{bmatrix} \tag{2.16}$$

Here the diagonal entries are representative of what was covered in the previous section. The superscript notation helps distinguish the use of multiple solution meshes. For instance the matrix $K_D^{\phi\phi}$ is the stiffness matrix assembled with basis

functions from only the ϕ mesh. Whereas the matrix $M_{\kappa}^{T\phi}$ is the mass matrix that is assembled using basis functions from the T and ϕ meshes. The off-diagonal terms are produced in the same way that the diagonal terms are produced, except that the test function and shape function are defined over independent meshes. If the two variables have the same spatial mesh, this detail is inconsequential. However, if the two variables have different spatial meshes, special care must be taken when assembling these terms; this situation is discussed in section 2.4.2

2.4 Spatial Adaptivity

2.4.1 Motivation

It can be shown that the convergence of the L^2 norm solution error ($\|u - u_h\|_{L^2}$) is dependent on the size of mesh cells used to approximate the solution with order $\mathcal{O}(h^{p+1})$, where h is the cell size and p is the order of polynomial approximation [23]. Equation 2.17 predicts the asymptotic error in the solution when compared to the spatial mesh size. It is sometimes more convenient to express this reduction in error in terms of the number of unknowns in the solution vector; Equation 2.18 describes the error convergence in this sense.

$$\varepsilon \propto h^{p+1} \tag{2.17}$$

$$\varepsilon \propto (N_{dofs})^{-\frac{p+1}{d}} \tag{2.18}$$

Equations 2.17 & 2.18 describe how the solution error will behave when the domain is refined uniformly. This type of refinement can yield very accurate results, but in most cases the computational load becomes prohibitive. Thus uniform refinement is reserved for diagnostic purposes to determine whether the numerical methods have been implemented correctly.

In general spatial refinement is needed in places where the local solution error is large. An error estimator can be used to locate where local solution error resides in the spatial domain; the exact error estimator for the Laplacian operator is known as the Kelly Error Estimator. This estimator determines the jump of the solution gradient across mesh cell boundaries [11]. This estimator is not an exact estimator for the equations studied in this Thesis, but the goal is not to know the error to a high accuracy only to locate those cells where the error is large compared to other locations in the domain. Thus the Kelly Error Estimator will be referred to as an error indicator in this sense because it cannot estimate the size of the numerical error, but only where the error is localized. The rule of thumb is that where the gradient is large, the spatial error is also large; thus refinement is likely to be needed where there are localized features.

The type of AMR studied in this Thesis is h-AMR where the "h" refers to the spatial mesh size. Other types of AMR involve varying the polynomial approximation degree and are given the letter "p" (p-AMR, hp-AMR) [23]. While these additional types of AMR could yield more accurate results, this Thesis concentrates on h-AMR since the results can easily include hp-AMR, if available. In all cases where AMR is used, the goal is to solve the problem at hand with the same precision from using a uniformly refined mesh with a lower computational budget. The savings in computational budget can either be used to solve the problem faster or more precisely.

2.4.2 Coupled Physics

This Thesis is concerned with solving coupled physics problems where each physics component potentially has different properties (i.e., local features in different parts of the domain). From the discussion in the previous section, it can be postulated that a

more optimal setting would have each physics component defined on an independent mesh. This means that each solution component can be free to have an optimized mesh that is independent of other variables.

A point that is necessary to discuss at this time is the implication of using Multimesh AMR on coupled physics problems. This technique may introduce issues if care is not taken to overcome them. In the coupling term for ϕ and T, the residual contains the following integral (Equation 2.19).

$$\left[\int_{\Omega} b_i^{\phi} \kappa b_j^{T}\right] \phi \tag{2.19}$$

When implementing these types of integrals in finite element software, this type of integral is transformed into a sum of integrals over the cells that make up the domain (as shown in Equation 2.15). This transformation is no longer possible since the basis functions for each variable is defined over different cells, i.e, b_i^{ϕ} is defined on $\mathcal{T}_{\phi}$ and b_j^{T} is defined on $\mathcal{T}_{T}$. The first observation of how to avoid this conundrum was made in the paper by Yaqi Wang [21]. In this paper, the authors introduce an algorithm to assemble the discrete version of Equation 2.19 based on the assumptions of:

1. All independent meshes are derived from the same initial coarse mesh
2. Refinement is achieved by regular bisection of cells
3. Embedded finite element spaces are used where each basis function on a cell can be represented as a linear combination of basis functions on the next refined cell.

Under the first two assumptions the meshes for each variable, though independent, are connected; there can always be a mesh found that is the intersection of the

two independent meshes, denoted $\mathcal{T}_\phi \cap \mathcal{T}_T$. Thus the integral in Equation 2.19 can be written as Equation 2.20.

$$\left[\sum_{K \in \mathcal{T}_\phi \cap \mathcal{T}_T} \int_K b_i^\phi \kappa b_j^T \right] \phi \tag{2.20}$$

For the intersection mesh $\mathcal{T}_\phi \cap \mathcal{T}_T$, at least one of the basis functions, b_i^ϕ or b_j^T, must be defined on the cell K, but the other could be defined on parents of the cell K. The terms "parent" and "child" cell are used to convey the way adapted meshes are produced; each cell from both meshes starts from the same state and, when refined, produces child cells which can be further refined. Since embedded finite element spaces are used on each cell, the parent cell's basis functions can be represented as a linear combination of the child cell's basis functions.

Assume that a cell in $\mathcal{T}_T$ is once more refined than a cell in $\mathcal{T}_\phi$; the cell in $\mathcal{T}_\phi$ (K_p) is a parent of the cells in $\mathcal{T}_T$ (K_c). Thus the basis functions of K_p need to be represented on K_c to complete the transformation in Equation 2.20. Equation 2.21 shows that a matrix can be constructed to interpolate basis functions defined on K_p to K_c.

$$b_i^\phi|_{K_c} = B_c^{il} b_l^T|_{K_c} \tag{2.21}$$

The matrix B_c^{il} interpolates data from a parent cell to its c^{th} child cell. Since a child of cell K_p can have children of its own, this process can be computed recursively until the cell on the intersection mesh (K) is reached. When all basis functions have representations on the cells of the intersection mesh, the integral over a cell K is evaluated using numerical quadrature.

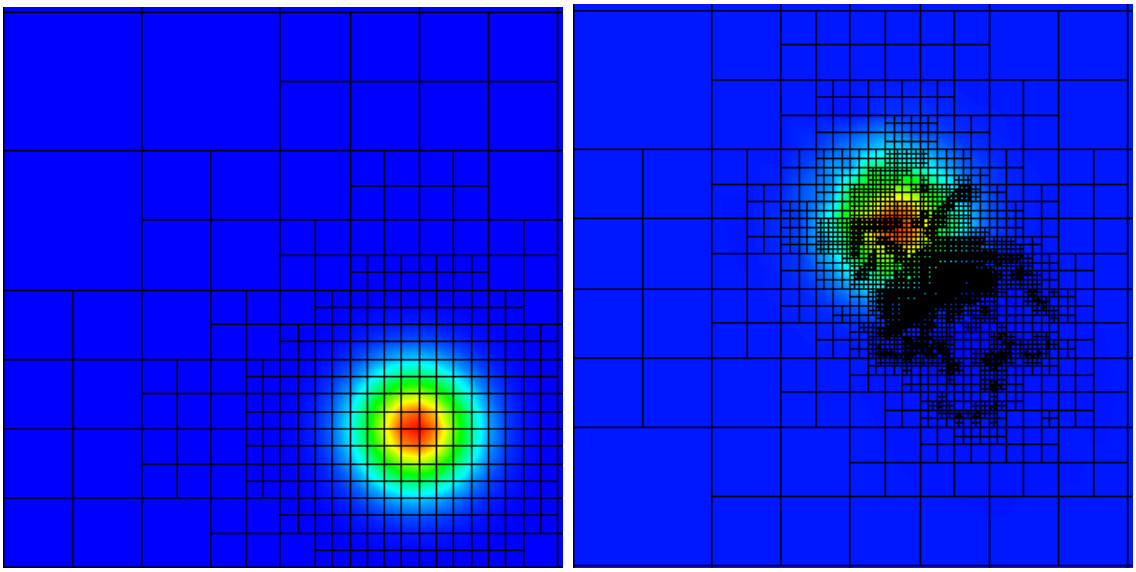
2.5 Dynamic Mesh Refinement

A consequence of using AMR for time dependent problems is that the spatial mesh may need to change in time. Previous publications on dynamic AMR [6, 13, 20] have performed mesh refinement at every time step. This strategy can become overly costly if the solution is not changing rapidly in time. Additionally, these algorithms start spatial refinement for each time step from the initial coarse mesh; this method is mathematically clean (each spatial mesh has no influence from previous meshes), but can become costly. For this Thesis, a modified version of the algorithm in [6, 13, 20] is implemented where each time step starts spatial adaptivity on the mesh from the previous time step. An additional feature is then implemented where the same mesh can be used for multiple time steps if the solution is not changing significantly. This added feature, referred to as the "Propagating Mesh", is designed to further balance the need for accurate solutions and shorter compute times.

2.5.1 Adaptivity at Every Time Step

A version of the algorithm presented in [6, 13, 20] is implemented in this Thesis, where the mesh from the previous time step is used as the starting mesh for the current time step's AMR process. The solution used to test this algorithm has a steep gradient and will force the cells around the solution to become small, as shown in Figure 2.2. To prevent either component of the solution error (spatial or temporal) from dominating, the two discretization sizes should remain on the same order ($h \sim \tau^{\frac{s}{p+1}}$). However, by utilizing this constraint, the decreasing mesh size would drive the time step size smaller and would eventually halt the simulation. The algorithm developed for this Thesis uses equally sized time steps and does not adhere to this constraint. Consequently, if the solution is forced to move past small spatial cells, oscillations can occur; this is observed in Figure 2.2 as the refinement "wake"

in the solution's path.



(a) t = 0s (b) t = 0.75s

Figure 2.2: Unlimited spatial mesh size under dynamic AMR

To prevent the type of refinement behavior in Figure 2.2, a limit on the mesh size is introduced. The mesh size control is implemented by not allowing a cell to be refined further than a set maximum level of refinement. The maximum level of refinement is chosen to be the same as the initial level of refinement input by the user; the user specifies how much to refine the initial mesh. The justification for this choice stems from the observation that any error in the initial condition will propagate through the time evolution. Thus, having a fine mesh on the final solution is not helpful unless the initial condition had the same level of spatial resolution.

The implementation of the spatial mesh size limit may not be the best for general problems, but is shown to work well with the problems tested. A more robust method would base the mesh size limiter on the time step size. This type of mesh size limit

is discussed in the section on future work. The next section discusses the feature added to the dynamic AMR algorithm.

2.5.2 Propagating Mesh Feature

An extension of the algorithm used in [6, 13, 20] can be to use the same spatial mesh over multiple time steps when the solution is slowly varying in time. To implement this extension, one needs some way to quantify the change of a solution in time and be able to judge when and where to spatially adapt. The methods described in this section address these concerns and utilize the mesh size control feature from the previous section.

During the spatial adaptation process, an indicator for the solution error is obtained on a cell-wise basis. This indicator forms a vector in $\mathbb{R}^n$ where n is the number of mesh cells. If the solution does not change in time, this vector is stationary; as the solution changes in time, this vector will deviate from its original state. The angle between the original vector and the current vector can be computed using the dot product (Equation 2.22). When the angle between the two vectors is large enough, the solution is said to have moved significantly and the spatial mesh needs to be adapted again.

$$\theta_i = \cos^{-1} \frac{V_1 \cdot V_i}{\|V_1\|\|V_i\|} \leq \theta_{\text{tol}} \tag{2.22}$$

Here V_1 & V_i are the error vectors after the last spatial refinement and at the current time, respectively; the index "i" denotes the current time step. The user can specify the acceptable angle (θ_{tol}) that triggers spatial refinement (usually between 1° - 45°). A schematic of this process is shown in Figure 2.3 where a given adapted spatial mesh is re-used over several time steps from time t_1 to time $t_{1+\alpha}$, with α denoting the number of times Equation 2.22 was satisfied.

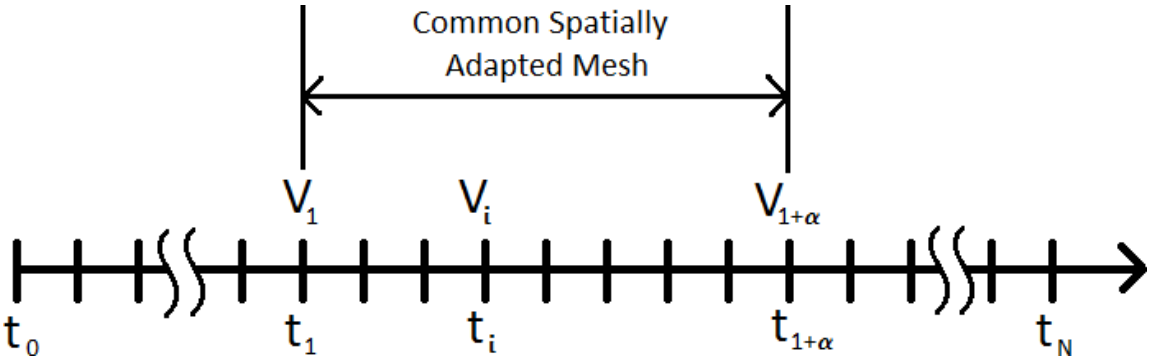


Figure 2.3: Time domain schematic when a spatial mesh is used for multiple time steps

Figure 2.3 shows that the same spatial mesh is used for $t \in [t_1, t_{1+\alpha}]$. In the schematic, "$i$" represents the current time step. Consequently, the parameter α is not known *a priori*, but only when $\theta > \theta_{\text{tol}}$ by using the vectors V_1 & $V_{1+\alpha}$. The vectors $\{V_i\}$ are stored, and when the spatial mesh needs refinement these vectors are averaged (Equation 2.23) to produce an error indicator for where the mesh should be adapted.

$$\overline{V} = \frac{1}{1+\alpha} \sum_{i=1}^{1+\alpha} V_i \tag{2.23}$$

The justification for determining the error indicator in this way comes from the idea that the spatial refinement should reflect any changes in the solution that happened during the time while the mesh was fixed. Averaging the error indicators over this time interval will produce a gross estimate of how the solution behaved in the interval $[t_1, t_{1+\alpha}]$.

An extension to the Propagating Mesh could be to restart the time evolution after each spatial adaptation. From Figure 2.3, when the solution reaches time $t_{1+\alpha}$, the mesh would be spatially adapted based on the error indicator in Equation 2.23

and the solution process restarted from time t_1. This extension was briefly studied during this Thesis and will be discussed in the future work section.

The solution techniques discussed in this chapter were implemented in code, using namely the Deal.ii Finite Element Library [1, 2]. The Deal.ii Library is a finite element library written in C++, widely used in academia and industry, and offers a high quality environment for the rapid development of software based on adaptive finite elements. Each method that is implemented in code needs to be tested to ensure that a mathematically correct answer is produced from the code. The next chapter discusses the ways to test the methods presented in this chapter, along with the testing results.

3. CODE VERIFICATION

In any application where a model of a physical process is conducted, the question always in mind is "How close is the result to reality?" This question can be broken into two other distinct questions, namely "How accurately does the model approximate reality?" and "How accurately does the implementation solve the model?" The first question is addressed through code validation, or the testing of a model with experiments to determine how closely the model can predict the experimental results [16]. Code validation can be an arduous task that involves a high amount of development and resources; this Thesis will not be concerned with the validation of the models employed here. However, the second question of code verification, or the testing of the implementation of the model, is addressed. Code verification requires thought and planning, but is a more manageable goal when compared with code validation.

The process of code verification finds a way to compare an exact solution to a computed solution. One way this can be achieved is to transform the problem under consideration into a simpler problem — one for which an exact solution exists. For example, the material properties could be made constant throughout the domain and observe how the computed solution behaves. Unfortunately, when making the problem simpler, one is not testing the problem in its most general sense and implementation errors can be overlooked. A more robust process would produce an exact solution without making the problem simpler. One such method, used in this Thesis, is known as the Method of Manufactured Solutions [17].

The Method of Manufactured Solutions (MMS) involves specifying an exact solution that will have interesting features to test. This solution is input into the

undiscretized governing equation and produces a residual, since it is unlikely that the exact solution will satisfy the original equation. This residual is added as a forcing term in the governing equation so that the specified exact solution will satisfy the equation. There are many advantages to this method; the most distinguished is that the problem being solved does not need to be modified, except for adding a forcing term.

In the following sections, several manufactured solutions are used. The problem parameters common to all manufactured solutions are given in Table 3.1 and the nonlinear form of the neutron absorption coefficient is given by Equation 3.1. In all of the cases, the domain length in all directions is $L = 10$.

Table 3.1: Problem Parameters Common to All Manufactured Solutions

ϕ	value	T	value	ϕ	value	T	value
Σ_a^0	3.0	γ	0.001	C_ϕ	2.5	C_T	1.6
$\nu(1-\beta)$	1.44	T_{ref}	3.0	x_0	5.0	x_0	-5.0
Σ_f	5.0	κ	0.001	y_0	5.0	y_0	-5.0
D	2.0	k	1.0	σ_ϕ	0.1	σ_T	0.1

$$\Sigma_a(T) = \Sigma_a^0 \left[\gamma(T^2 - T_{\text{ref}}^2)\right] \tag{3.1}$$

For the types of problems tested in the following sections, the material properties ($\Sigma_a, D, \kappa, \gamma$, etc.) are inconsequential since they are used to produce the residual for MMS. The parameters that determine the solution behavior are the manufactured solution parameters (C_ϕ, C_T, x_0, y_0, etc.).

3.1 Verification of Spatial Discretization

A good starting point for code verification is to determine whether the steady state problem is being solved correctly. Rothe's method is used to discretize the temporal domain, which is essentially solving a steady state problem at every time step [3]. Thus, if the steady state problem is not being solved correctly, there is no hope that the transient problem will be solved correctly.

The metric used to determine whether the spatial solution is being solve correctly or not is to compare the solution error ($\|u - u_h\|_{L^2}$) to the mesh size. In Section 2.4, Equation 2.17 showed that the error decreased at a prescribed rate as the spatial mesh size decreased. Thus the solution error will be computed for meshes with a decreasing cell size, and the solution error should trend as in Equation 2.17 after a sufficiently small mesh size is reached.

The choice of solution is an important aspect of MMS. It is desirable to have a solution that cannot be exactly represented using a finite polynomial; since the approximation space is spanned by polynomial functions. If the same degree of polynomial is used for the approximation space as the exact solution, the spatial error will be close to machine precision and spatial refinement would not further decrease the solution error. Thus an exact solution to test spatial convergence could be of a trigonometric or exponential form in space, for instance.

The proposed manufactured solution to test the spatial convergence is a "stationary Gaussian peak". The functional form is given by Equation 3.2 where the exponential provides the Gaussian peak, and the quadratic term specifies the Dirichlet boundary values. This equation describes the solution to a single solution component; each solution component can have different peak positions, amplitudes, and peak widths. A visual representation of one component of the solution is shown in

Figure 3.1.

$$U(\vec{x}) = C_u \prod_{i=1}^{d} \left[1 - \left(\frac{x_i}{L} \right)^2 \right] e^{-\frac{\left(x_i - x_i^{0,u} \right)^2}{\sigma_u}} \tag{3.2}$$

The variable x_i denotes a spatial coordinate, while $x_i^{0,u}$ denotes the location of the peak for component "u". In this manufactured solution the parameter x^0 does not change in time, making the peak stationary.

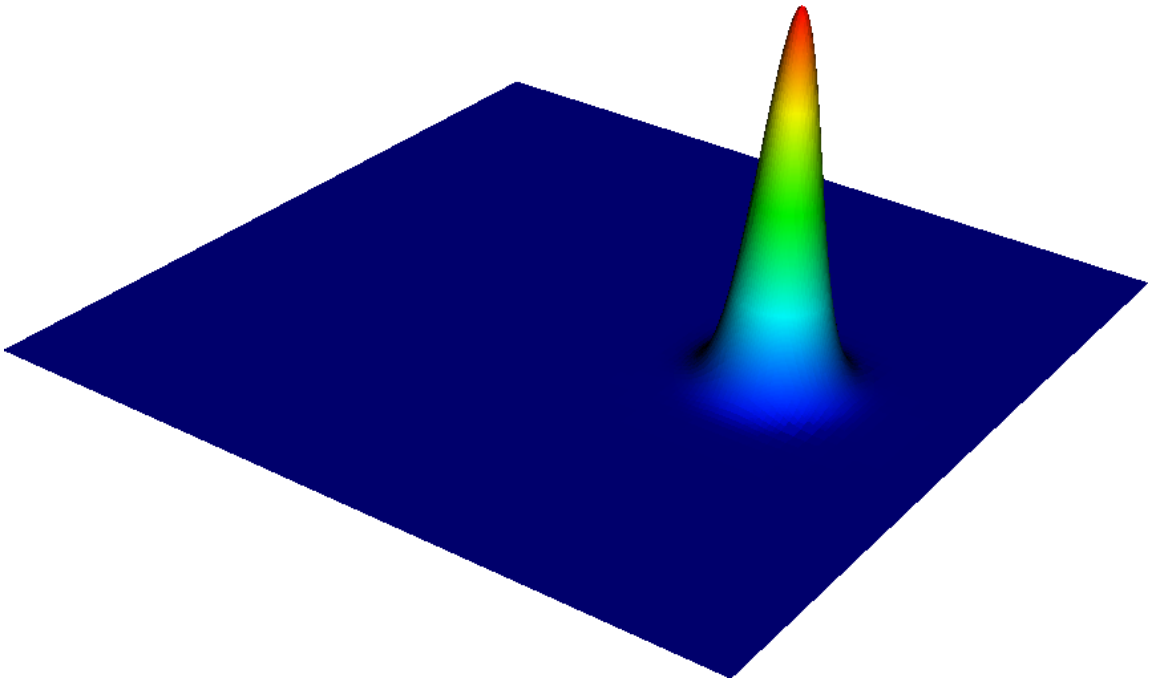
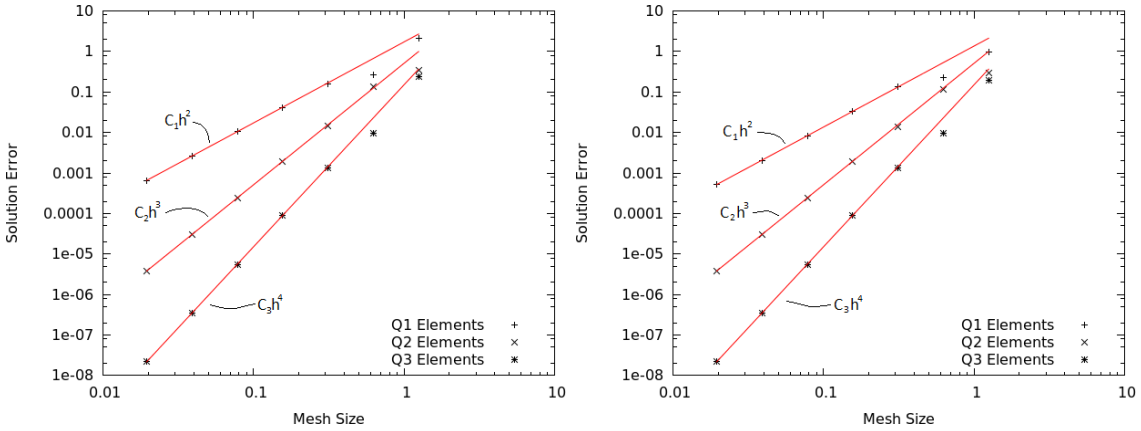


Figure 3.1: Stationary Gaussian manufactured solution for one of the two solution components. Solution for ϕ is shown.

The rate of decrease in the solution error depends on both the mesh size and the order of the polynomial approximation, as was seen in Equation 2.17. In Figure 3.2, the solution error is evaluated for decreasing mesh sizes; the computation is repeated

for Q1, Q2, and Q3 elements. The slopes denoted in the figure are the expected convergence rates from Equation 2.17.



(a) ϕ (b) T

Figure 3.2: Spatial convergence for uniform mesh refinement

Observe that as the spatial mesh size decreases, the solution error also decreases at the predicted rate after a sufficiently small mesh size. When the mesh size is still large ($\mathcal{O}(1)$) the solution error does not yet decrease at the predicted rate. This can be attributed to the spatial mesh being to coarse to accurately represent the solution and thus the convergence rate only holds if the spatial mesh represents the solution well enough. In slightly different words: the convergence rate asymptotically approaches predicted convergence rate as $h \to 0$. The solution error appears to decrease according to the predicted rate, and thus the spatial discretization is declared to be implemented correctly.

3.2 Verification of Temporal Discretization

Once the steady state problem is determined to be implemented correctly, the transient problem can be addressed. In the same way as the spatial component, it

is desirable to isolate the time portion of the solution. An excellent choice for a manufactured solution to test the transient problem would have very little spatial error so that the majority of the solution error was produced by the time integrator. In the previous section it was noted that if the manufactured solution is a polynomial in space and the approximation finite element space spans polynomials of at most the same degree, then the spatial error would be close to machine precision; this is the type of behavior desired here to isolate the temporal error.

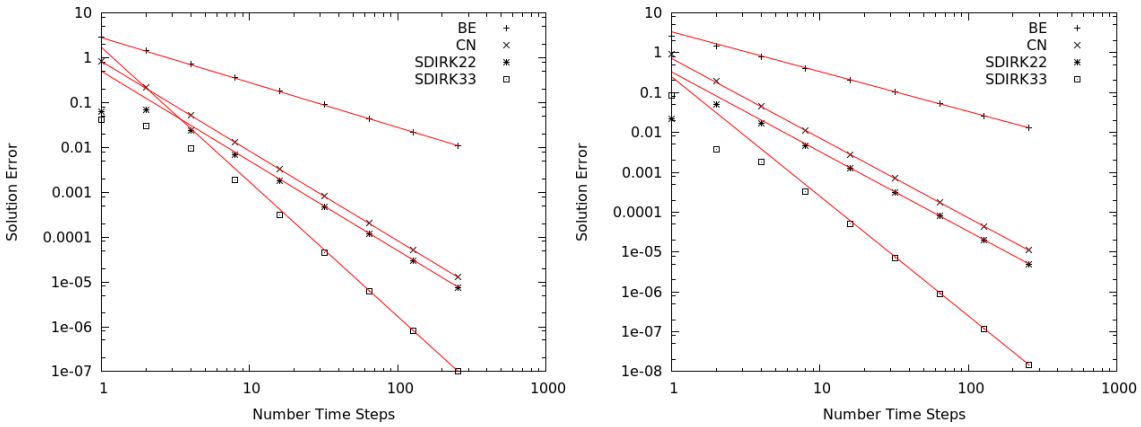
To test the time integrator, the manufactured solution's spatial component is chosen to be some polynomial with the same degree as the approximation space. The time portion of the manufactured solution makes the solution change in some way that cannot be represented as a polynomial, so that the temporal behavior cannot be resolved exactly by the time integrator. Various time integrator methods are tested that were included in Table 2.1, each having a specific convergence rate. The proposed manufactured solution is given by Equation 3.3 where the solution is a quadratic "bubble" function in space with a temporal amplitude changing as a sine wave. Each solution component's amplitude changes at a different rate ($\omega_\phi = 1.5, \omega_T = 0.2$).

$$U(\vec{x}, t) = C_u \left[\sin(\omega_u t) + 1\right] \prod_{i=1}^{d} \left[1 - \left(\frac{x_i}{L}\right)^2\right] \tag{3.3}$$

The amplitude can vary between zero and two; in the simulations presented the end time is chosen to be less than $\frac{3\pi}{2}$ so that the solution is always non-zero. The solution being zero should not detract from the results obtained since there is nothing in the models that prevent a variable from being zero.

To determine whether the time integrator is working properly, much like in the previous section, the transient problem is computed with various sized time steps

and the solution error should converge at a rate proportional to the time integrator order. The test is set up so that the spatial approximation uses Q2 elements that will resolve the spatial component to within machine precision, and the time interval is $t \in [0, 1s]$. Since the time interval is fixed, as the number of time steps increases the size of the time step decreases; in fact there is a one-to-one relationship between the number of time steps and the time step size.



(a) ϕ (b) T

Figure 3.3: Temporal convergence for several time integrator methods

Figure 3.3 shows the solution error compared to the number of time steps taken. Four time integrator methods were tested that give convergence rates ranging from linear to cubic. For both variables, the solution error decreases at the predicted rate and thus it seems that the temporal portion of the implementation is correct.

3.3 Connection between Space and Time

In reality, the spatial error and the temporal error both contribute to the overall solution error. In general the overall solution error will converge as $\mathcal{O}(h^{p+1} + \tau^s)$, where h and τ are the spatial mesh size and the temporal step size respectively. Both

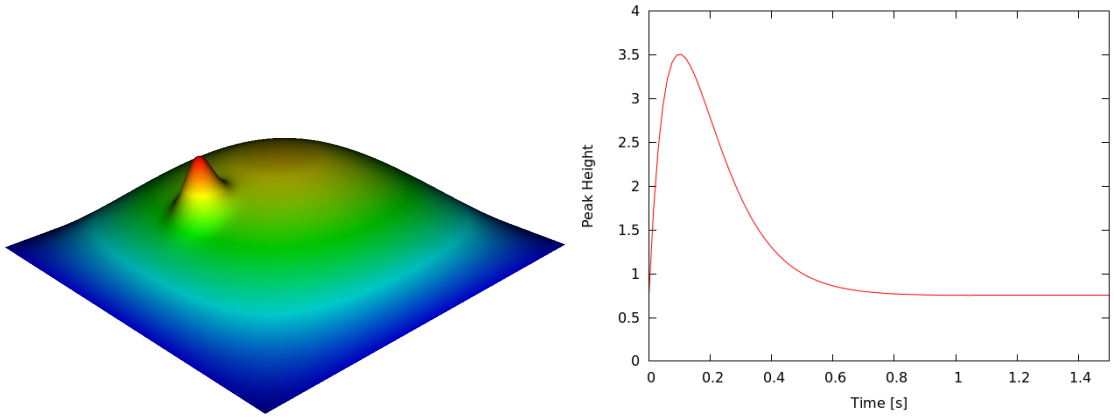
parameters $\{h, \tau\}$ must be controlled relative to each other; if one is large compared to the other, that error will dominate the solution error. In the previous sections, the size control was addressed by only allowing a specified level of refinement for a cell. A reasonable approach would be to limit the spatial mesh size as $h \sim \tau^{\frac{s}{p+1}}$, since the time step size is fixed in these simulations.

To show that the spatial mesh size and the time step size are connected, a manufactured solution is devised that cannot be resolved by a discrete polynomial approximation in either space or time. The proposed solution has a smooth and quadratic initial condition much like the solution described in the time verification section. After the simulation starts, a Gaussian peak grows into the solution. The height of the Gaussian peak grows as a Maxwellian in time so that the initial height is zero and grows to a maximum and then decays exponentially. The form of the manufactured solution is given by Equation 3.4. The behavior of this solution is similar to what would happen in a nuclear reactor during a rod ejection accident; the initial flux is smooth, and when the accident occurs a local spike in the flux appears and then the entire solution renormalizes. Because of the similarity to a reactor accident, this solution will be referred to as the "Reactor" solution in this section. In each of the solution components, the peak is stationary ($\omega_{\phi,T} = 0.0$) and the growth factor is moderate ($\alpha_{\phi,T} = 1.0$).

$$U(\vec{x}, t) = C_u \left[\prod_{i=1}^{d} \left[1 - \left(\frac{x_i}{L} \right)^2 \right] + \frac{\alpha_u}{\sigma_u} t e^{-\alpha_u t} \prod_{i=1}^{d} \left[1 - \left(\frac{x_i}{L} \right)^2 \right] e^{-\frac{\left(x_i - x_i^{0,u} \right)^2}{\sigma_u}} \right] \tag{3.4}$$

If α_u is made large, the peak will appear and disappear rapidly; also if σ_u is made small, the peak will occupy a small region of the domain. The constant, $\frac{\alpha_u}{\sigma_u}$, ensures that the height of the peak will be much greater ($\sim 10\times$) than the maximum of the

initial condition. A graphical representation of the manufactured solution, described by Equation 3.4, is shown in Figure 3.4. After the simulation begins the localized peak quickly grows, reaches a maximum, and then decays away exponentially.



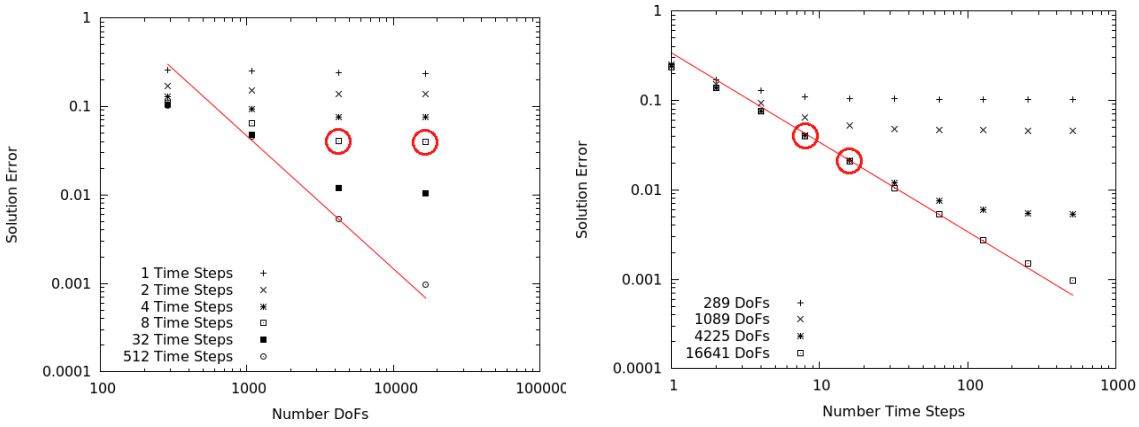
(a) Manufactured solution visual representation

(b) Time behavior of solution at peak location (5,0)

Figure 3.4: Manufactured solution with smooth initial condition and appearing peak. Solution for ϕ is shown.

Figure 3.5 shows the convergence of the solution when using Q2 spatial elements and the Implicit Euler time integrator with $t \in [0, 1s]$. Figure 3.5a shows how the solution error decreases as the spatial grid is refined for different numbers of time steps. It is seen that for a low number of time steps, the spatial convergence plateaus; this is caused from the time integrator error dominating the solution error. As the number of time steps increases and the temporal error no longer dominates, the theoretical spatial convergence rate is approached. Likewise Figure 3.5b shows the temporal convergence for differing levels of spatial refinement. The same effect is observed in that if the spatial resolution is coarse, the temporal convergence rate plateaus due to the spatial error dominating. As the mesh is made finer and the spatial error no

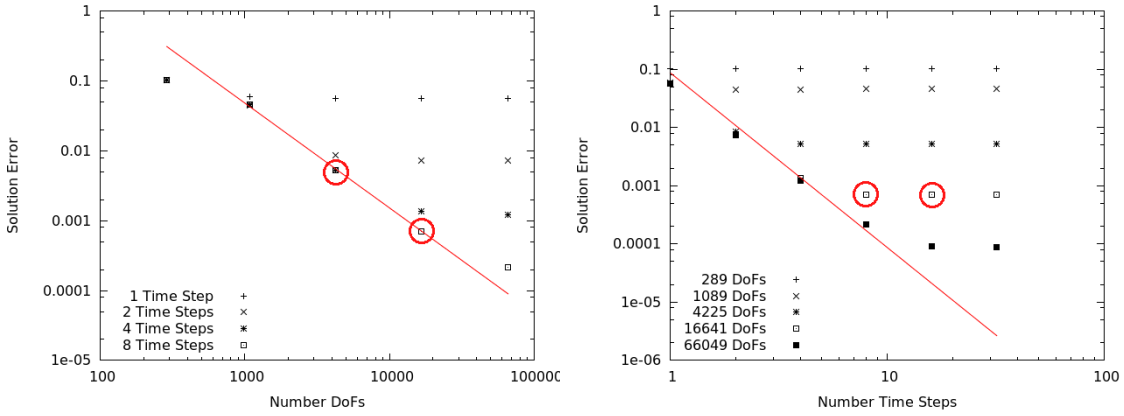
longer dominates, the temporal convergence approaches the theoretical rate.



(a) Spatial convergence (b) Temporal convergence

Figure 3.5: Convergence with space and time using BE time integrator and Q2 elements

Figure 3.6 shows the convergence for the same problem, but instead of Implicit Euler an SDIRK33 time integrator is used. Since this time integrator is a higher order, it is expected that the temporal component of the solution will be more resolved than is the case of a low order method with the same time step size.



(a) Spatial convergence (b) Temporal convergence

Figure 3.6: Convergence with space and time using SDIRK33 time integrator and Q2 elements

Comparing Figure 3.5a and Figure 3.6a shows that the spatial convergence rate can be achieved in many fewer time steps when a higher order time integrator is used. The highlighted points in Figures 3.5a & 3.6a correspond to the same spatial and temporal discretizations. Notice that in Figure 3.5a, the solution error only decreases slightly between the two points. Conversely in Figure 3.6a, the solution error decreases at the predicted rate. This result appeals to intuition because as the time integrator order increases, the temporal error will be reduced and the spatial error convergence rate should approach the result presented in Figure 3.2.

Comparing Figure 3.5b and Figure 3.6b shows that a much more refined mesh must be used to obtain the correct temporal convergence rate. Again since the temporal error is reduced by increasing the integrator order, the spatial resolution must increase so that it will not dominate the solution error; this is exactly what is observed in transitioning from Figure 3.5b to Figure 3.6b. In order to produce the correct temporal convergence rate, the spatial resolution must be increased. The goal of this discussion was to emphasize the importance of requiring a connection between

the spatial mesh size and the temporal step size in the form of a discretization size limit as discussed in Section 2.5.2.

3.4 Properties of Propagating Mesh

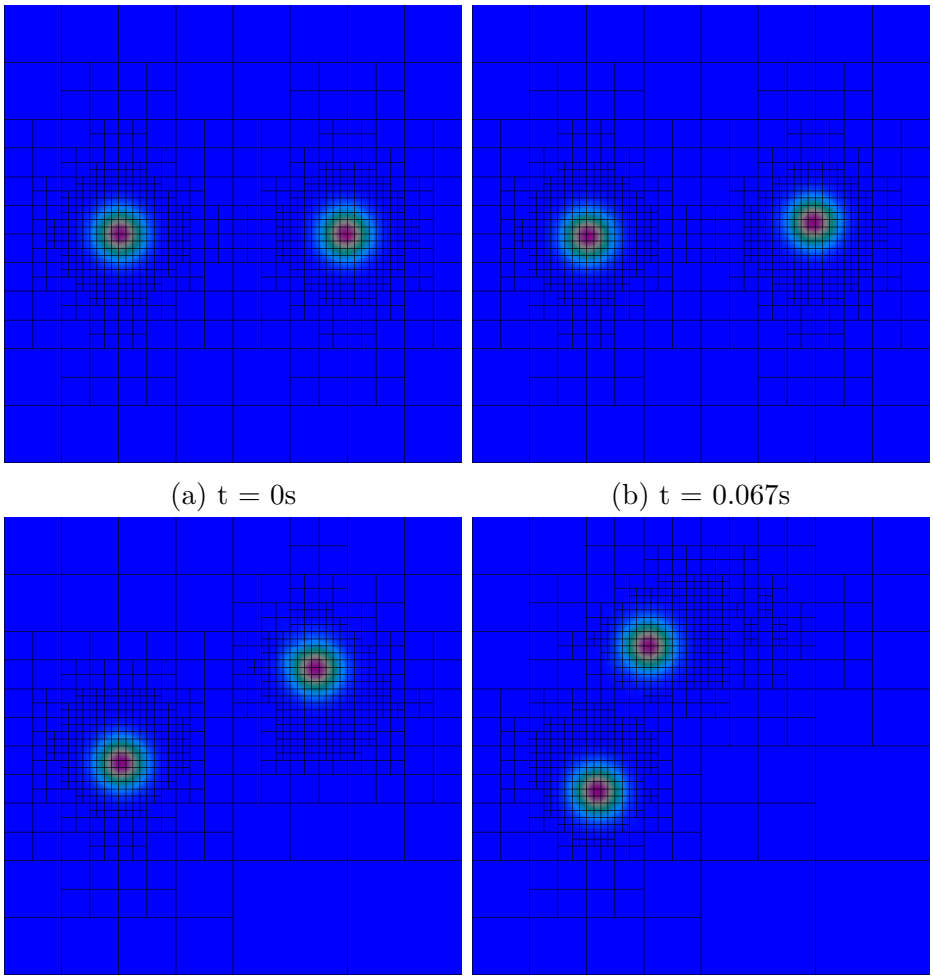
The propagating mesh feature was tested in two ways by manufactured solutions that had different properties. The first solution is given by Equation 3.2 except that the Gaussian peak is rotating around the origin so that x^0 is given by Equation 3.5. The values of the rotation speed for each solution component is the same as the temporal discretization verification ($\omega_\phi = 1.5, \omega_T = 0.2$). This solution tests the propagating mesh algorithm by having the peak present in the initial condition so that the initial spatial mesh will be refined in a concentrated region of the domain. Thus, the spatial mesh will be required to travel along with the Gaussian peak during the time evolution.

$$x^{0,u}(t) = \begin{bmatrix} sin(\omega_u t) \\ cos(\omega_u t) \\ 0 \end{bmatrix} \tag{3.5}$$

The second solution is given by Equation 3.4 again. This solution is initially smooth an thus the refinement is scattered throughout the domain; a Gaussian peak emerges from the smooth solution and decays away. The propagating mesh algorithm should concentrate refinement around the peak even if refinement is not present in the initial condition.

Figure 3.7 shows four snapshots in the evolution of the first solution referred to as the "Traveling Gaussian" solution. Several observations can be made about how the algorithm handles this type of solution. First, Figure 3.7a shows the initial condition for this solution. The mesh is refined in a concentrated region around the

peak and is relatively coarse in the other parts of the region. There is however, a radius of refinement that is larger than the solution radius; this can be attributed to a constraint, imposed by the finite element software used [1, 2], to have only a single hanging node per edge. This constraint actually allows for an advantageous feature that the solution can move in this refined region without requiring the mesh to be refined further; in Figure 3.7b the solution has moved slightly, but the mesh is still exactly the same.



(a) t = 0s (b) t = 0.067s

(c) t = 0.5s (d) t = 1.33s

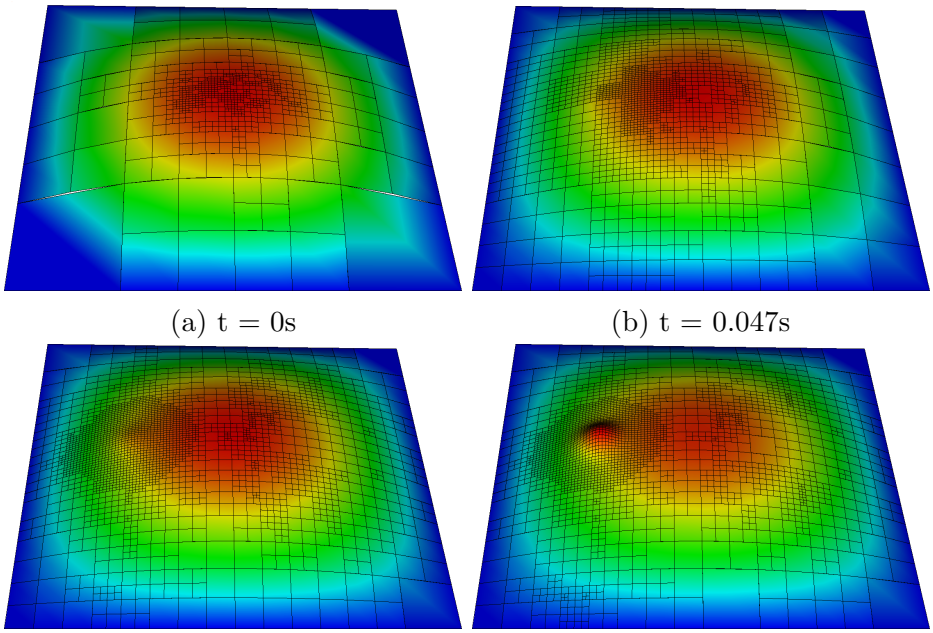
Figure 3.7: Traveling Gaussian solution with dynamic AMR

As time progresses, the mesh lags behind the solution. The lagging is caused by the fact that the error vector used to determine which cells need refinement is an average vector of each time step's error vector since the last refinement (Equation 2.23); naturally, where the solution had been in the past would also have a large error.

As opposed to Figure 2.2, this figure shows no wake in the solution path. This is attributed to the spatial size limit, which was not present in Figure 2.2. In addition, the mesh that is behind the solution is coarsened as the solution moves away from that region so that under utilized Degrees of Freedom (solution unknowns) can be freed.

Figure 3.8 shows four snapshots in the evolution of the second solution referred to as the "Reactor" solution. This solution differs from the Traveling Gaussian solution in that the initial condition is smooth and thus the initial mesh is not able to anticipate where refinement will be needed.

Figure 3.8a shows the initial condition for this solution, and the refinement seems to be sporadically placed close to the center of the domain. As the Gaussian peak begins to emerge in Figure 3.8b, the refinement drifts from the center of the domain to the left half of the domain. By the time Figure 3.8c is reached, the peak is visible to the eye and the mesh is concentrated around the peak much like in Figure 3.7. After this point the mesh is more or less static except for a few small perturbations. Figure 3.8d is near the end of the time interval for this solution; the peak is clearly visible and will start to decrease in magnitude from this time forward. The mesh is not expected to change back to the initial state because an exponential is never zero and thus the peak will always be present.



(a) t = 0s (b) t = 0.047s

(c) t = 0.094s (d) t = 0.914s

Figure 3.8: Reactor solution with dynamic AMR

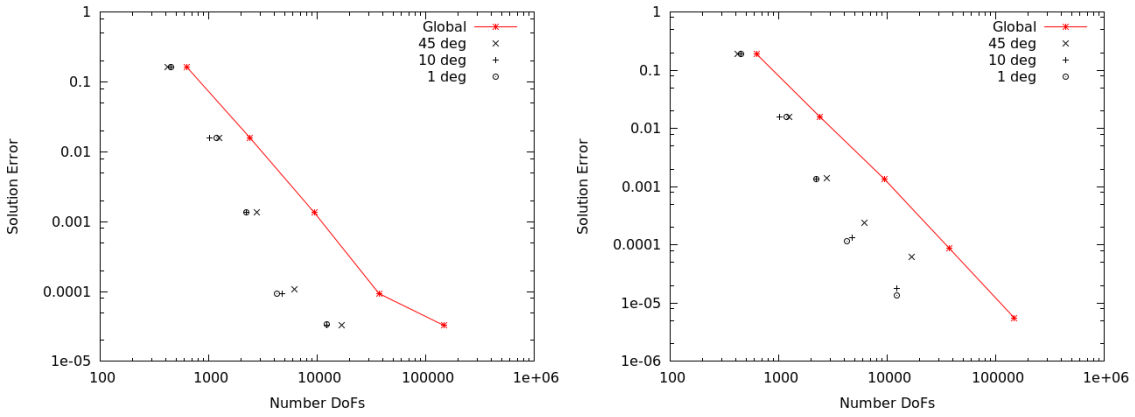
This section has been a qualitative analysis of how the propagating mesh will handle certain types of problems. The next section provides a quantitative analysis of how much better the propagating mesh can be when compared to conventional techniques of uniformly refined meshes and refinement at every step.

3.5 Performance of Propagating Mesh

In this section, the same manufactured solutions analyzed in the previous section are computed using a third-order time integrator (SDIRK33) with 128 time steps in the interval $t \in [0, 1s]$. Three angle tolerances (45°, 10°, 1°) are compared against a static uniformly refined mesh with the metrics of "Number of DOFs" (memory consumption) and "CPU Time" (computation time).

Figure 3.9 shows the solution error versus problem size (Number DOFs) for both solution variables. For each variable, the adapted mesh always produces a smaller

system to solve, and gives an accurate result. The result is not surprising since the spatial mesh for the Traveling Gaussian solution is mostly coarse except for a concentrated area of the domain. There seems to be close agreement between the 1° & 10° tolerances. However, the 45° tolerance seems to begin to plateau; this can be an indicator that 45° is not a well chosen tolerance. Also, note that in Figure 3.9a the convergence rate starts to plateau similar to what was seen in the discussion from the previous section. This is an indication that the temporal discretization's contribution to the solution error is beginning to dominate and that for this level of refinement, the time step size is too large.

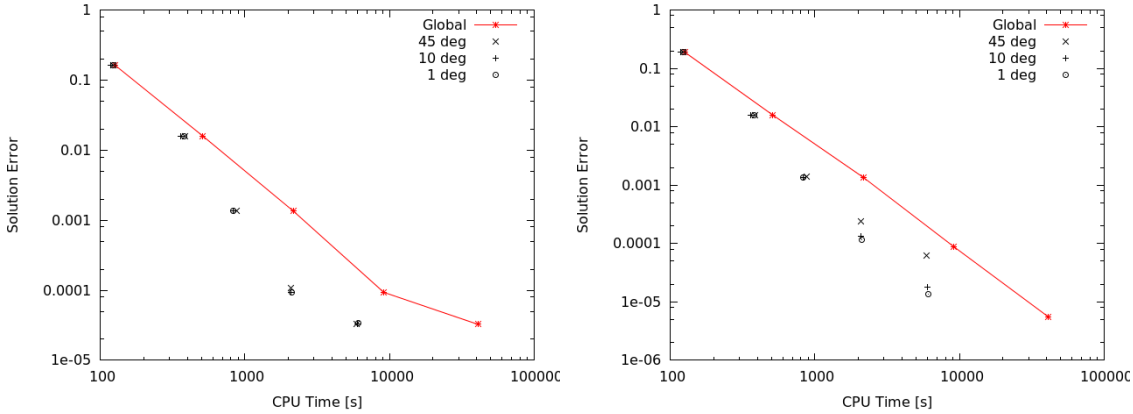


(a) ϕ (b) T

Figure 3.9: Convergence of Traveling Gaussian solution vs. system size

Figure 3.10 consists of the same set up, but the solution error is compared against the run time. Incidentally the added time to construct the system of equations on the adaptively refined mesh is not more than the time required to solve the larger system of the uniformly refined mesh. Again this result is not surprising since the size of the uniformly refined system is roughly 10× as large as the adaptively refined

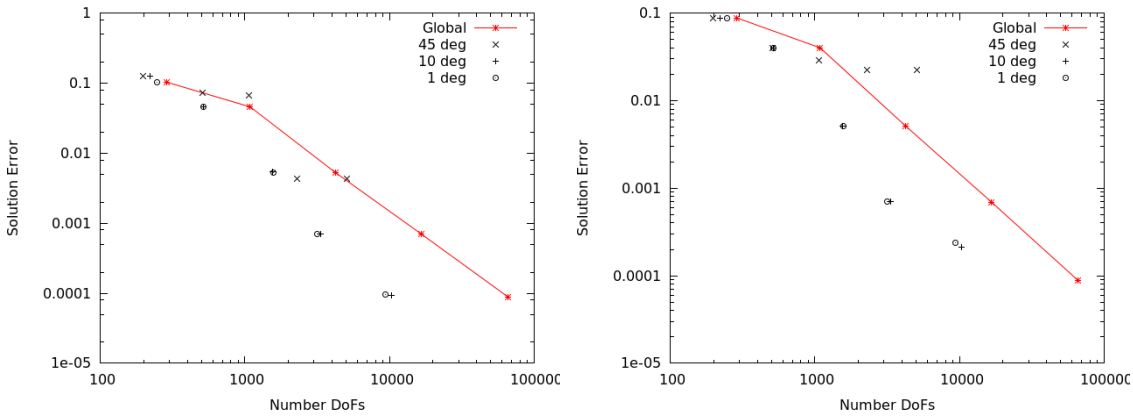
systems. Even with the added complexities of hanging nodes and coupling terms, the adaptively refined meshes seem to win.



(a) ϕ (b) T

Figure 3.10: Error of Traveling Gaussian solution vs. run time

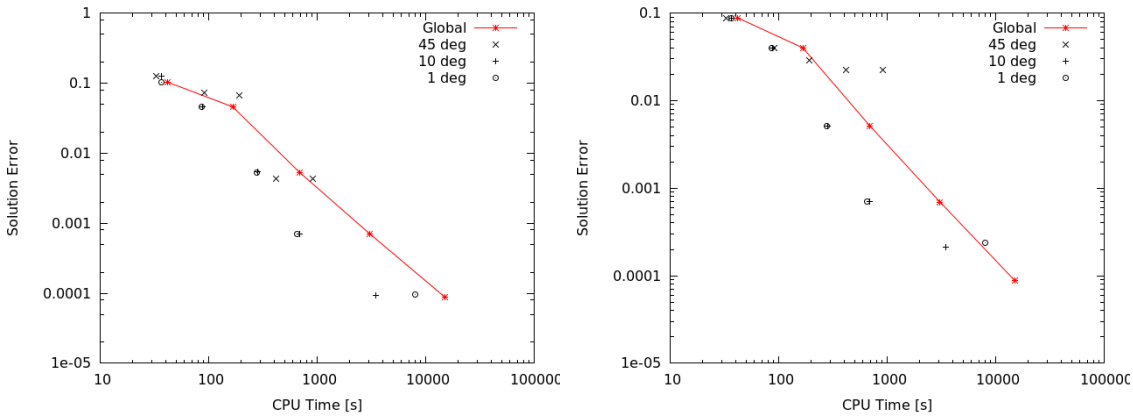
Switching to the Reactor solution in Figure 3.11, the same type of behavior is observed. What is now more pronounced is that the 45° tolerance is not fine enough to resolve the changing solution. Especially for the second variable in Figure 3.11b, the 45° tolerance plateaus very early. By contrast, the 1°&10° tolerances seem to coincide well with each other.



(a) ϕ (b) T

Figure 3.11: Convergence of Reactor solution vs. system size

Figure 3.12 shows the same problem set, but compared against the run time. The results are not as clear as in the previous cases; it seems that after a certain point, the refined mesh with a 1° tolerance takes just as long to compute as the uniformly refined static mesh. Since the size of the adapted system is close to 10× smaller than the uniformly refined system, the increase in time must come from the assembly process and not the linear solver. This result is not very reassuring to previous algorithms since the 1° tolerance would be faster than the methods presented in the literature. From the computational time results, the propagating mesh algorithm seems to produce a solution faster than previous algorithms would.



(a) ϕ (b) T

Figure 3.12: Error of Reactor solution vs. run time

Overall, the algorithm for propagating a spatial mesh across several time steps seems to utilize the computational resources more efficiently than previously proposed algorithms would. It is shown that an acceptable value for θ_{tol} is close to 10°. The resources freed from this algorithm can either be used to solve the problem more accurately or in a faster time.

4. CONCLUSIONS

4.1 Applicability of AMR

The results presented in the previous sections show that using Multimesh AMR, on various problems that are of interest in nuclear engineering, can be advantageous. The advantages appear in the size of numerical system that is required to be solved, and the time required to reach a solution. In every case tested, the system constructed using Multimesh AMR was about $10\times$ smaller than the system constructed using a globally refined static mesh. In addition, the time required to assemble the system constructed by Multimesh AMR does not, in general, take longer than the savings from solving a smaller system.

The techniques explored through this Thesis are general enough that they could be adapted to other applications in nuclear engineering. One area where the propagating mesh could make an impact would be in the study of shock propagation through material. This application quantifies how discontinuities in material states move through a domain, similar to the behavior of the Traveling Gaussian solution. While the Traveling Gaussian solution does not posess the same properties as a shock front (i.e., the Gaussian solution is infinitely differentiable, while a shock front is not), the need for spatial refinement to move through the domain as the solution moves through the domain is similar in both solutions. The propagating mesh algorithm could potentially refine the spatial mesh around the shock front as it moved through the domain.

4.2 Dynamic AMR

The philosophy driving Adaptive Mesh Refinement is to produce a spatial mesh that is optimally suited for the specific solution. This philosophy can be extended

to the temporal dimension in that the spatial mesh should change in time with the solution. Many of the articles presented in the introduction use a strategy of spatially refining at every time step. Additionally the refinement stopping criteria is based on being able to produce an accurate error estimator, which might not be available; instead, the strategy presented in this Thesis relies on the temporal behavior of an error indicator which is easily obtained. This strategy allows for a lightweight version of the strategies presented in the literature for the dynamic refinement of spatial meshes.

A concern for the propagating mesh strategy was that it would be dependent on the initial condition of the solution. Since the behavior of nuclear reactors during accidents (an important facet of nuclear engineering) has solutions which are initially smooth and develop a large localized discontinuity, this algorithm needs to be able to handle this situation. The Reactor manufactured solution was devised to address this concern directly.

Based on the results presented in the previous chapter, this method is not dependent on the initial condition. Moreover, there is no reason to believe that there would be a dependence on the initial condition. Since the method measures how the error changes spatially, if a new feature to the solution is added there should be a large change in the error at that location. This method seems to be an appropriate method for solving the types of problems found in the nuclear engineering field.

4.3 Lessons Learned

With the completion of any project, it is imperative to reflect on what revelations were conceived during the project; this step allows for the revelations to be applied to future projects. During the course of this Thesis, many important lessons were learned and two of the most influential will be discussed presently.

At the Graduate level, the concept of non-stationary linear solvers are introduced. While these solvers work very well for large matrix systems, they are generally not as efficient for small matrix systems as direct methods. An enticing trap to fall into as a young Graduate student is to use the non-stationary solvers in all cases. The first lesson learned during this Thesis was that when solving small matrix systems ($n \lesssim 10,000$), a direct solver will outperform an iterative solver. A large amount of time was spent at the beginning of this project waiting on the linear solver to converge until a direct solver was finally implemented.

The concept that the error from both spatial discretization and temporal discretization contribute to the overall solution error seems obvious. However, when performing convergence analysis on the solution error, this phenomena became painfully obvious. It was first discovered when testing the temporal convergence of the traveling Gaussian solution with a low solution speed; since the solution speed was low, the temporal error could be resolved quite well. The temporal convergence rate began to plateau when the time step size was made smaller. It became obvious to the author that if the temporal error could be resolved well, then the spatial error would contribute more to the total error; the hypothesis was tested with a positive outcome, by making the spatial mesh smaller. The second lesson learned during this Thesis was that the spatial mesh size and the temporal step size need to balance each other. One cannot simply reduce a portion of the error and expect that the overall solution error will behave as expected.

4.4 Future Work

As with any project, there can always be improvements. This section outlines a few areas that could be expanded upon in the future.

This Thesis focused on spatial adaptivity, but another equally important concept

is temporal adaptivity. If a solution is changing rapidly in time, a smaller time step size would be desired to better resolve the solution's behavior. However, if the solution is slowly varying a larger time step size can be taken to avoid unnecessary computations. Temporal adaptivity can be used to more efficiently utilize computational resources, just as in spatial adaptivity. A class of time integrator methods that are able to estimate the error introduced by the temporal discretization are Embedded Runge-Kutta (ERK) methods. The implementation of ERK methods could also be used to obtain a more appropriate θ_{tol}. The error obtained at each time step remaining low (indicating that the solution is slowly varying in time) would trigger the use of a larger θ_{tol}, while the error remaining large would trigger the use of a smaller θ_{tol}. There might still be heuristics involved in determining the relationship between the ERK error and the size of θ_{tol}, but the heuristics would be more sophisticated.

In the presented dynamic mesh strategy, the spatial mesh size was limited by the number of initial refinements input by the user. This input is generally based on a heuristic estimate of how the solution will behave. In reality the spatial mesh size is only limited by the size of a time step. Moreover, the total solution error behaves as $\mathcal{O}(h^{p+1} + \tau^s)$ and thus to keep both parameters balanced the relation $h \sim \tau^{\frac{s}{p+1}}$ should hold. This could serve as an excellent limit on the spatial mesh size instead of a heuristic input. Additionally, if temporal adaptivity were implemented the mesh size limit would adjust to the time step size without user input.

In the quest for more accurate simulation techniques, the size of the system is likely to increase. In many cases, the memory consumption for the Jacobian matrix used in Newton's Method will become prohibitive. In such cases it is beneficial to use low-storage methods such as the Jacobian-Free Newton Krylov (JFNK) method. These methods use the fact that non-stationary linear solvers do not actually require access to the system matrix, but only require the action of the matrix on a given

vector. Thus, a numerical approximation of the Jacobian can be made using the perturbation from vectors that span a Krylov subspace; using JFNK then only requires the residual to be stored.

An extension to the Propagating mesh algorithm could be to restart the solution process after spatial discretization. This extension would, in essence, send trial solutions to test how the solution will change; restarting the solution after spatial adaptation allows the AMR process to gather information on what the spatial mesh will need to be. Preliminary tests on this extension were conducted and produced solutions closer to what would be computed on a uniformly refined mesh. For the problem tested the gain in accuracy was not drastic, but for other types of problems the results could be different.

REFERENCES

[1] W. Bangerth, R. Hartmann, and G. Kanschat. `deal.II` – a general purpose object oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4):24/1–24/27, 2007.

[2] W. Bangerth, T. Heister, and G. Kanschat. `deal.II` *Differential Equations Analysis Library, Technical Reference*, 2012. `http://www.dealii.org`.

[3] I. Blank and P. Smith. Convergence of rothe's method for fully nonlinear parabolic equations. *The Journal of Geometric Analysis*, 15:363 – 372, 2005.

[4] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons - West Sussex, England, 2nd edition, 2008.

[5] K. T. Clarno, B. Philip, W. K. Cochran, R. S. Sampath, S. Allu, et al. The amp (advanced multiphysics) nuclear fuel performance code. *Nuclear Engineering Design*, 252:108 – 120, 2012.

[6] L. Dubcova, P. Solin, J. Cerveny, and P. Kus. Space and time adaptive two-mesh hp-fem for transient microwave heating problems. *Electromagnetics*, 30(1 – 2):23 – 40, 2010.

[7] J. Guermond and A. Ern. *Theory and Practice of Finite Elements*. Springer-Verlag - New York, 2004.

[8] J. D. Hoffman. *Numerical Methods for Engineers and Scientists*. Marcel Dekker - New York, 2nd edition, 2001.

[9] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM - Philadelphia, 1995.

[10] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*. SIAM - Philadelphia, 2003.

[11] D. W. Kelly, J. P. Gago, O. C. Zienkiewicz, and I. Babuska. A posteriori error analysis and adaptive processes in the finite element method: Part I Error analysis. *International Journal for Numerical Methods in Engineering*, 19:1593–1619, 1983.

[12] D. A. Knoll, A. K. Prinja, and R. B. Campbell. A direct newton solver for the two-dimensional tokamak edge plasma fluid equations. *Journal of Computational Physics*, 102(2):424 – 425, 1992.

[13] D. Lebrun-Grandié, J. Ragusa, and B. Turcksin. Adaptive multimesh hp-fem for a coupled neutronics and non-linear heat conduction problem. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, 2011.

[14] R. B. Lowrie. A comparison of implicit time integration methods for nonlinear relaxation and diffusion. *Journal of Computational Physics*, 196(2):566 – 590, 2004.

[15] A. Mezzacappa and O. E. B. Messer. Neutrino transport in core collapse supernovae. *Journal of Computational and Applied Mathematics*, 109(1–2):281 – 319, 1999.

[16] W L. Oberkampf and T. G. Trucano. Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences*, 38:209 – 272, 2002.

[17] P. J. Roache. Code verification by the method of manufactured solutions. *Transactions of the ASME*, 124:4 – 10, 2002.

[18] L. Segerlind. *Applied Finite Element Analysis*. John Wiley & Sons - Hoboken, NJ, 2nd edition, 1984.

[19] P. Solin, J. Cerveny, L. Dubcova, and D. Andrs. Monolithic discretization of linear thermoelasticity problems via adaptive multimesh *hp*-fem. *Journal of Computational and Applied Mathematics*, 234(7):2350 – 2357, 2010.

[20] P. Solin, L. Dubcova, and J. Kruis. Adaptive *hp*-fem with dynamical meshes for transient heat and moisture transfer problems. *Journal of Computational and Applied Mathematics*, 233(12):3103 – 3112, 2010.

[21] Y. Wang, W. Bangerth, and J. Ragusa. Three dimensional h-adaptivity for the multigroup neutron diffusion equations. *Progress in Nuclear Energy*, 51:543–555, 2009.

[22] Y. Wang and J. Ragusa. Application of hp adaptivity to the multigroup diffusion equations. *Nuclear Science and Engineering*, 161:22–48, 2009.

[23] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *Finite Element Method — Its Basis and Fundamentals*. Butterworth Heinemann - London, 6th edition, 2005.

[24] S. P. Zwart, S. McMillan, S. Harfst, D. Groen, M. Fujii, et al. A multiphysics and multiscale software environment for modeling astrophysical systems. *New Astronomy*, 14(4):369 – 378, 2009.

APPENDIX A

BUTCHER TABLEAUX

Using a general s-stage Runge-Kutta method allows many types of time integrators to be implemented with ease. To change between methods, only the values and sizes of the matrices in a butcher tableau are changed. A general Butcher Tableau has the form below with A being a matrix and b, c being vectors.

c	A
	b^T

In general the order of the method determines the size of the matrices. All methods implemented in this thesis are diagonally implicit methods. The first method implemented is the Implicit Euler (Backward Euler) method which is first order.

Backward Euler

1.0	1.0
	1.0

The first of the second order methods implemented is the Crank-Nicholson method, here is its Butcher Tableau. The Crank-Nicholson sequence gives even weighting to the first and second halves of a time step.

Crank-Nicholson

0	0	
1	0.5	0.5
	0.5	0.5

The next of the second order methods is the SDIRK22 method (Singly Diagonally Implicit Runge-Kutta); the 22 stands for 2 stages and 2$^{\text{nd}}$ order.

SDIRK22

$$\begin{array}{c|cc} \gamma & \gamma & \\ \sigma+\gamma & \sigma & \gamma \\ \hline & \sigma & \gamma \end{array}$$

$$\gamma = \frac{2 \pm \sqrt{2}}{2} \qquad \sigma = 1 - \gamma$$

The only third order method implemented is SDIRK33. The parameter γ is a root of the equation given below the Butcher Tableau.

SDIRK33

$$\begin{array}{c|ccc} \gamma & \gamma & & \\ \frac{1+\gamma}{2} & \frac{1-\gamma}{2} & \gamma & \\ 1 & \frac{-6\gamma^2+16\gamma-1}{4} & \frac{6\gamma^2-20\gamma+5}{4} & \gamma \\ \hline & \frac{-6\gamma^2+16\gamma-1}{4} & \frac{6\gamma^2-20\gamma+5}{4} & \gamma \end{array}$$

$$\frac{1}{6} - \frac{3}{2}\gamma + 3\gamma^2 - \gamma^3 = 0 \qquad \gamma = 0.435866521508459$$